第七章　第七次作业

1. 在 MySQL 中，存储过程和存储函数的主要区别在于：

   A. 存储过程可以返回单个值，而存储函数可以返回数据集

   B. 存储函数必须指定返回值类型，而存储过程不需要

   C. 存储过程可以包含 DDL 语句，而存储函数不可以

   D. 存储过程可以使用 CALL 语句调用，而存储函数不能

   怎么感觉第一题的 B 和 C 都对呢。

2. 在创建存储过程时，以下哪个参数类型是不允许的?

   A. IN　　B. OUT　　C. INOUT　　D. ALL

3. 以下哪个 SQL 语句用于创建一个存储过程，该过程接受一个整数类型的输入参数并返回一个包含单个整数值的输出参数?

   A. `CREATE PROCEDURE my_procedure (IN num INT) BEGIN ... END;`

   B. `CREATE PROCEDURE my_procedure (OUT num INT) BEGIN ... END;`

   C. `CREATE FUNCTION my_function (IN num INT) RETURNS INT BEGIN ... END;`

   D. `CREATE PROCEDURE my_procedure (INOUT num INT) BEGIN ... END;`

4. 以下哪个 SQL 语句正确地在存储过程中使用了 IF 语句?

   A. `IF condition THEN BEGIN ... END;`

   B. `IF condition THEN BEGIN ... END IF;`

   C. `BEGIN IF condition THEN ... END;`

   D. `IF condition THEN ... END IF;`

5. 以下哪个 SQL 查询将正确地返回每个部门的员工数量?

   A. `SELECT Department, COUNT(*) FROM Employees GROUP BY Department;`

   B. `SELECT COUNT(*) FROM Employees GROUP BY Department;`

   C. `SELECT Department, COUNT(*) FROM Employees ORDER BY Department;`

   D. `SELECT Department FROM Employees GROUP BY Department;`

6. 已知 tm_employee 表中具有默认约束 df_adress，删除该约束的语句为

A. `alter table tm_employee drop constraint df_adress;`

B. `alter table tm_employee remove constraint df_adress;`

C. `alter table tm_employee delete constraint df_adress;`

D. `remove constraint df_adress from talbe tm_employee;`

7. 使用 bank 数据库，查询每个分行的员工总数以及该分行所有员工管理的客户数量。

   示例：CUST_ID 为 10 的客户的账户 24，其 OPEN_EMP_ID 为 16，则该客户是员工 16 管理的客户。

```
SELECT
  branch.NAME,
  COUNT( * ),
  sum( a.cust_num )
FROM
  employee
  LEFT JOIN branch ON employee.ASSIGNED_BRANCH_ID = branch.BRANCH_ID
  LEFT JOIN ( SELECT OPEN_EMP_ID, COUNT( * ) as cust_num FROM account GROUP
  ↪  BY OPEN_EMP_ID ) as a on a.OPEN_EMP_ID = employee.EMP_ID
GROUP BY
  BRANCH_ID;
```

NAME	员工总数	客户数量
上海市总行	9	8
建国支行	3	7
南京分行	3	3
杭州分行	3	6

8. 使用 bank 数据库，找到每个分行的员工中直接下级员工最多的员工，输出字段包括员工 id，全名，所属分行名，直接下级数量。

   注意：直接下级指，如果员工 A 的 SUPER_EMP_ID 是 B，那么 A 是 B 的直接下级员工。

```
-- 主要是要考虑可能出现多个相同的最大值（虽然这里没有），所以使用了三层嵌套子查
↪  询。
SELECT
  employee.EMP_ID as 员工id,
  CONCAT( employee.LAST_NAME, employee.FIRST_NAME ) as 全名,
  branch.`NAME` as 所属分行名,
  ( SELECT COUNT( * ) FROM employee as a WHERE a.SUPERIOR_EMP_ID =
  ↪  employee.EMP_ID ) as 直接下级数量
FROM
```

```
  employee
  JOIN (
    SELECT
      ASSIGNED_BRANCH_ID,
      MAX( 直接下级数量 ) as 最大直接下级数量
    FROM
      ( SELECT ASSIGNED_BRANCH_ID, ( SELECT count( * ) FROM employee as a
      ↪  WHERE a.SUPERIOR_EMP_ID = employee.EMP_ID ) as 直接下级数量 FROM
      ↪  employee ) as a
    GROUP BY
      a.ASSIGNED_BRANCH_ID
  ) as b USING ( ASSIGNED_BRANCH_ID )
  LEFT JOIN branch ON employee.ASSIGNED_BRANCH_ID = branch.BRANCH_ID
WHERE
  ( SELECT COUNT( * ) FROM employee as a WHERE a.SUPERIOR_EMP_ID =
  ↪  employee.EMP_ID ) = 最大直接下级数量;
  --   最外层好像不用 group by 也能用 having，但是会多出一列，还是直接 where 吧，
  ↪  但是 where 不能引用前面计算出来后用 as 命名的
```

员工 id	全名	所属分行名	直接下级数量
4	李易枫	上海市总行	5
10	陈易	建国支行	2
13	蒋琴琴	南京分行	2
16	杨天宝	杭州分行	2

9. 使用 bank 数据库，查询该银行每个部门的员工管理的账户中，余额最少的账户的类型，输出字段为部门名，账号产品类型，余额。

```
SELECT
  department.`NAME` as 部门名,
  product_type.`NAME` as 账号产品类型,
  account.AVAIL_BALANCE as 余额
FROM
  ( SELECT employee.DEPT_ID, MIN( AVAIL_BALANCE ) as min_balance FROM
  ↪  employee JOIN account ON employee.EMP_ID = account.OPEN_EMP_ID GROUP BY
  ↪  DEPT_ID ) as a
  JOIN employee USING ( DEPT_ID )
  JOIN account ON employee.EMP_ID = account.OPEN_EMP_ID
  left JOIN product USING ( PRODUCT_CD )
  LEFT JOIN product_type USING ( PRODUCT_TYPE_CD )
  LEFT JOIN department USING ( DEPT_ID )
WHERE
  AVAIL_BALANCE = min_balance;
```

部门名	账号产品类型	余额
营业部	存款	1057.7500
营业部	存款	1057.7500
综合管理部	存款	125.6700

先处理多个最值的情况，也有固定套路，先在子查询里查询出最值，运行一下看看没问题了，只保留聚合的键（DEPT_ID）以及最值（AVAIL_BALANCE），再 join 上原来的表（employee, account），并对最值使用 where 子句过滤（where AVAIL_BALANCE = min_balance），就完成了多个最值的处理。之后保留此表的信息，所以 left join 其他几个表，检查一下没问题了最后使用投影函数也就是 select 语句指定想要获取的列。

10. 使用 bank 数据库，查询每个客户（包括个人和企业）的账户中，交易金额最高的单笔交易信息

    要求：显示客户全名、账户 ID、交易 ID、交易金额和交易日期。

```
SELECT
  ( CASE customer.CUST_TYPE_CD WHEN 'i' THEN CONCAT( individual.LAST_NAME,
  ↪  individual.FIRST_NAME ) ELSE business.`NAME` END ) as 客户全名,
  ACCOUNT_ID AS 账户ID,
  TXN_ID as 交易ID,
  AMOUNT as 交易金额,
  TXN_DATE as 交易日期
FROM
  ( SELECT CUST_ID, MAX( AMOUNT ) as max_amount FROM acc_transaction RIGHT
  ↪  JOIN account USING ( ACCOUNT_ID ) GROUP BY CUST_ID ) as a
  JOIN account USING ( CUST_ID )
  JOIN acc_transaction USING ( ACCOUNT_ID )
  LEFT JOIN customer USING ( CUST_ID )
  LEFT JOIN individual USING ( CUST_ID )
  LEFT JOIN business USING ( CUST_ID )
WHERE
  AMOUNT = a.max_amount;
```

客户全名	账户 ID	交易 ID	交易金额	交易日期
尤青	2	2	500000.0000	2011-01-15 00:00:00
许文强	5	5	2000000.0000	2012-03-12 00:00:00
何婕	8	14	650000.0000	2013-12-15 00:00:00
吕东	11	3	650600.0000	2011-01-15 00:00:00
施珊珊	13	19	340023.0000	2015-01-27 00:00:00
张晓	14	9	3330000.0000	2013-08-24 00:00:00
孔庆东	17	18	5000.0000	2015-01-12 00:00:00
曹方	19	7	15000.0000	2012-05-23 00:00:00
严匡	23	23	15000.0000	2015-06-30 00:00:00
华东师范大学	25	11	503292.0000	2013-10-01 00:00:00
阿里巴巴集团有限公司	27	21	119345.0000	2015-03-22 00:00:00
上海汽车集团股份有限公司	28	16	30000.0000	2014-07-30 00:00:00
南瑞集团有限公司	29	20	200000.0000	2015-02-22 00:00:00

11. 使用 university 数据库，使用 with 子句进行递归查询，查询所有课程的所有先修课程列表（包含该课程）。

    例如：BIO-301 课程的所有先修课程列表为 BIO-101,BIO-301。

    注意：由于 prereq 表字段长度只有 varchar(8)，你可能需要使用 CAST 修改查询结果的类型。

    注：university 数据库在资料-上机材料中，其中数据库结构为 university.sql，smallRelationsInsertFile.sql。

    插入测试数据

    insert into prereq values('FIN-201', 'HIS-351');

    insert into prereq values('MU-199', 'FIN-201');

```
INSERT INTO prereq VALUES('FIN-201', 'HIS-351');
INSERT INTO prereq VALUES('MU-199', 'FIN-201');
WITH recursive full_prereq_table as (
( SELECT course.course_id, cast(course.course_id as VARCHAR(100)) as
↪  full_prereq_id FROM course WHERE not EXISTS ( SELECT course_id FROM
↪  prereq WHERE prereq.course_id = course.course_id ) ) UNION ALL
( SELECT prereq.course_id, concat(full_prereq_table.full_prereq_id, ",",
↪  prereq.course_id) FROM prereq, full_prereq_table WHERE prereq_id =
↪  full_prereq_table.course_id )
)
 SELECT * FROM full_prereq_table;
```

course_id	full_prereq_id
BIO-101	BIO-101
CS-101	CS-101
HIS-351	HIS-351
PHY-101	PHY-101
BIO-301	BIO-101,BIO-301
BIO-399	BIO-101,BIO-399
CS-190	CS-101,CS-190
CS-315	CS-101,CS-315
CS-319	CS-101,CS-319
CS-347	CS-101,CS-347
EE-181	PHY-101,EE-181
FIN-201	HIS-351,FIN-201
MU-199	HIS-351,FIN-201,MU-199

12. 使用 university 数据库，使用函数迭代查询某课程的所有先修课程列表（包含该课程），函数输入为一个代表课程 id 的字符串，输出为该课程的所有先修课程列表。

    例如：BIO-301 课程的所有先修课程列表为 BIO-101,BIO-301。

    插入测试数据

    insert into prereq values('FIN-201', 'HIS-351');

    insert into prereq values('MU-199', 'FIN-201');

```
DROP FUNCTION IF EXISTS get_prereq_courses;

CREATE FUNCTION get_prereq_courses(base_course_id VARCHAR(8)) RETURNS
↪  VARCHAR(100)
BEGIN
  DECLARE result VARCHAR(100);
  DECLARE child_temp VARCHAR(100);
  SET result = '';
  SET child_temp = base_course_id;
  WHILE child_temp is NOT NULL DO
    IF result = '' THEN
        SET result = child_temp;
    ELSE
        SET result = CONCAT(child_temp, ',', result);
    END IF;
    SELECT GROUP_CONCAT(prereq_id) INTO child_temp FROM prereq WHERE
    ↪  FIND_IN_SET(course_id,child_temp) > 0;
  END WHILE;
RETURN result;
```

```
END;

SELECT get_prereq_courses('BIO-301');
```

get_prereq_courses('BIO-301')
BIO-101,BIO-301

```
SELECT get_prereq_courses('MU-199');
```

get_prereq_courses('MU-199')
HIS-351,FIN-201,MU-199